Devanagari Character Recognition Using Artificial Neural Network

Vasu Negi [1], Suman Mann [2], Vivek Chauhan [3]
[1,2,3]Department of Information Technology, Maharaja Surajmal Institute of Technology,
Guru Gobind Singh Indraprastha University, New Delhi, India
[1] vasunegi@gmail.com
[2] sumanmann2007@gmail.com
[3] vivek.chauhan700@gmail.com

***Abstract*—Devanagari is one of the Ancient Scripts that is in regular use in India as well as Nepal. Devanagari is currently used by more than 120 languages including Hindi, Nepali, Marathi, and Pali which defines the prevalence and the domination of Devanagari [1]. Immense popularity of this script must be taken care by the advance technologies of our present world so that we are able to connect to the real world to a greater depth. Artificial Neural Network is one the fastest growing technologies that has attracted a lot of attention in the field of Machine Learning. Various new technologies have developed to implement fast neural networks with little in depth knowledge requirements. We will be using Keras along with Theano which are python libraries for building our neural network. In this paper we constructed a simple artificial neural network using keras to recognize isolated Devanagari characters and also analyze the impact of variations in parameters in learning phase.**

Keyword- Devanagari, *Character Recognition, Keras, Theano, Python*

I. INTRODUCTION

Devanagari script is one of the oldest script that is regular use in India as well as some other south Asian countries. Devanagari script is used in languages and dialects including many major languages such as Hindi, Marathi and Nepali [1]. The world is currently moving towards digitization and thus we require means to translate documents written in Devanagari into a digital format so that they can be preserved, manipulated and shared seamlessly.

Devanagari can be found to be used intensely in various areas of India from handwritten applications for installing a water connection or may be a letter of request to any concerned authorities, Devanagari is used by millions of Indians every day. Millions of documents are created written in Devanagari script every day, these handwritten documents must be taken care. Hence, we require efficient and flexible ways for doing so.

Artificial Neural Network is one of the technique that can be applied to efficiently recognize the Devanagari characters. Artificial Neural Network is simply a network of interconnected nodes that provides classification and regression abilities to the machine.

II. ARTIFICIAL NEURAL NETWORK

Artificial neural network are simply a network of interconnected nodes that provides classification and regression capabilities to the machine. These nodes are supposed to mimic biological neurons in a normal brain. Same as biological neurons, nodes does not have any computation on its own rather they act as a group of linear functions. Each node has the task to throw an output when a certain level of threshold has been received from other nodes [3].

The behaviour of each node can be defined by activation function which specifies. Activation function takes several inputs that comes from other nodes that are connected to a node and produces an output if a threshold is reached. A single node in itself cannot possess any computation capability. An ANN comprise of several nodes that are arranged in layers where each layer can comprise of several nodes. Each node in a layer is connected to another node in the adjacent layer.

A simple neural network with 3 layers comprises of an input layer, hidden layer and an output layer. The input layer does the task of collecting the input and the output layer presents the result in the form of classification using one of the several output nodes. Each node in the output layer will act as an independent class which will classify the data into one of the classes. Hidden layer is the layer where the learning takes place.

Each connection between nodes are assigned ‘strengths/weights’, usually assigned in a random fashion initially. These strengths attributed to each connection forms the basis of learning. These strengths are then incremented or decremented in way that the network is able to tune the output to a certain degree. A weight can also become zero denoting that the connection does not hold any importance.

We will be using Keras and theano which are python libraries for building our neural network. These libraries are discussed in depth in later sections.

III.LITERATURE REVIEW

In 1940, Donald Hebb, introduced a theory where he postulated that machines can learn by mimicking the behaviour of neurons. He focused that learning process must be dependent upon the strength between the interconnection of nodes. These strengths signify the importance of connection between those neurons in the network as a whole [4].

In 1957, Frank Rosenblatt introduced perceptron algorithm which is basically a learning algorithm for binary classification. This can be effectively used in simple pattern classification. Perceptron can be considered as a simple neural network where learning is based upon strengths between neurons. The principle behind the perceptron learning is how the weights are updated according to the output. The network starts with weights initialized randomly, if the output is wrong and has some deviation from the target output than weights are updated accordingly. But if correct the weights are not modified. The major problem is that perceptron is a linear classifier cannot converge when the training data set is not linear separable [5].

In 1982, Werbos described an algorithm known as back propagation. As we require the weights between the neurons to be updated we need some error to calculate the change required in the weights. It is an algorithm which defines how the error related information must be propagated backwards from the outer neuron to the inner layer. It defines a basis of how the error must be split along each of the link [6].

In 1998, Peter L. Bartlett, proposed that when the number of training data is significantly smaller than the number of weights, then the performance of the artificial neural network gets affected. His results showed that the performance of generalization can be influenced more by the number of the strengths when compared to the number of training instances [7].

In 2003, authors in [8], described the process for automatic recognition of vehicle license plate number He used image fusion technique to obtain a clearer image from a series of blurred images. The second step was to detect the license plate location in the image. Iterative thresholding can be used to locate the license plate. Several objects are recognized and labelled, these objects are then investigated and figured out whether they resemble a license plate or not. The next step is to extract the characters from the object which resembles the license plate. For recognizing the characters of a multilayered neural network was used which was trained using images of alphabets and digits. For classification purposes, images were added with additional noise to show the effectiveness of the process. The LPR system based upon neural network was able to achieve an accuracy of 95 percent.

In 2012, authors in [9], proposed a new technique to recognize image using ANN. The first step in the technique is to take a grayscale image & apply noise to it. Then for removing the noise they used adaptive median filter. This process converts the pixel into an 8-bit number. Now, 4 pixels are combined to form a number of 32 bits & convert it into a decimal. This process is repeated for each image. Now for recognizing an image, first the image is applied with noise & adaptive medium filter is used. Now the filtered image is compared with original image & average error is calculated. If the calculated error is greater than 45 percent then the image does not image else the image is a match. The overall accuracy was measured on the whole test data set in the same way. They contemplated that the ANN is able to recognize images with less average error when compared to the techniques not utilizing ANN.

Authors in [10], proposed a paper in which they tried to train a convolution neural network for classifying 1,200,000 images into 1000 unique classes. They developed a network with 5 convolution layers, 3 fully connected layers and a 1000-way softmax. Within the network, they also used pooling layers after the first and second convolution layers. In both convolution layer and fully connected layer they applied ReLU (Rectifier linear unit). The 1000-way softmax layer does the task of classifying the input to one of the 1000 classes. This neural network achieved an error rate of 15.3 percent which was quite significant.

IV. IMPLEMENTATION TOOL

A. GIMP

We used GIMP v2 for creating images of each character. GIMP stands for GNU Image Manipulation Program which is a cross-platform open source image editing software available for windows, Linux, OS X and other operating systems including android.

B. NUMPY

Numpy is a python package which provides various functions for scientific computations.Theano provides integration with Numpy which allows the use of various functions powered by Numpy into our code. It provides various features which are not available in Numpy.

C. THEANO

Theano is a python library that provides us with ways to compute and manipulate multi-dimensional arrays while taking into consideration effectiveness and efficiency. As theano is based on python which inherently

makes theano effective and agile providing a framework to quickly implement and manipulate large, complex and resource hungry mathematical expressions[11].

Theano provides the ability to perform computation on GPU(Graphic Processing Unit) which makes it much faster than computing on CPU. Our most latest CPUs are slow when compared on the basis of parallel computation, GPUs on the other hand are intensely fast as they does not work serially rather they dominate at parallel computing utilizing 100s of GPU cores.

One of the core features of theano is that it is very stable and prioritize optimization which makes it highly suitable for complex calculations environment. Theano can be used in most modern high computation intensive applications such as scientific research. It also provides the feature to detect various types of bugs with deep analysis of the error for better debugging.

D. KERAS

Keras is a neural network application programming interface(API) that is designed to work over Theano and TensorFlow. It is a python library which provides it the flexibility and power for fast implementation and testing of neural network. One of the characteristic of this API is that it provides user with easy to understand functions along with various options to choose from to finely tune the network.[12]

Keras is based upon modularity, modules can be finely tuned and glued together to construct a neural network model. This model can then be executed and tested efficiently without the need to worry over minute details. Each function can be configured to user's need, she can choose the activation function, loss function, optimizer and other parameters. Keras is supported by various major operating systems including Mac OS X, Linux and Ubuntu.

V. IMPLEMENTATION

A. STEPS

One of the important thing to mention is that the images that we created are in grayscale. The steps for creating the images area as follows:

1. Creating an image of 300x300 pixels.
2. Rescaling the image to 28x28 pixels.
3. Convert the image into 1D array of floating points.
4. Store the array of data in an excel(.csv) sheet.

The first step of creating our image was to choose the appropriate size of the original character so that images are easier to draw and less area for making mistakes. Let's suppose if we take the original size of image to be 720x720 pixels then there would be chances of making errors while making straight lines and making the curves which we know are the most important ingredient of this script. Or suppose if we take the original size of the image too small then it would be virtually impossible to take care of all the attributes of the character. Hence, we took the original size of the image to be 300x300 pixels.

Now we have the image of the character but it is too large as we will be working on each pixel and considering an image of size 300x300 that means we will have 90,000 pixels to work upon which is gargantuan as it will require huge amount of computation power and time. Thus we require the image to be shrunken down to a smaller size while trying not to neglect the amount of information in each image.

The second step is to rescale the image to a smaller size. The original image of size 300x300 was rescaled to 28x28 pixels. This reduces the total number of pixels to be accountable for further processing and analysis.

Now save this image in PNG format. The python libraries help us out in decoding the PNG files effectively.

After repeating the above process for creating multitude of images of different characters, we must progress to the third step which is converting the images into a suitable format. Our program does not work directly with images which are inherently 2D array of pixels where each pixel can have value within certain limits.

What we need to do is to convert the images into an array of values (floating point numbers). One of the python libraries that is scipy.misc allows us to do this conversion easily. We used a short python program to do the above mentioned conversion. This separate python program takes each image from our dataset and converts the image into a 1D array of floating point values and stores the data in an excel sheet (a file with .csv extension).

The first column of every image data defines the actual class the image belongs to and the rest of 784 columns defines the image data which is flattened into a 1D array. But the values will still be varying from 0 to 255 which is not acceptable as our program will work upon values ranging from 0 to 1. Hence we need to divide each 784 values of each image data by 255.

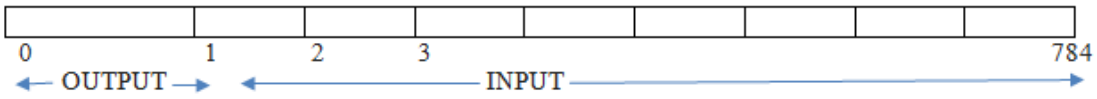

Fig. 1. Representation of image data in .csv file

B. CONSTRUCTION PHASE

For developing our neural network we will be using KERAS library with Theano backend. Our neural network will be working on the excel sheet (.csv file) which we created. We already know Keras is a high level API which allows us to implement neural network in a fluid way and theano on the other side is a python library which provides this API the ability to work on multi-dimensional arrays. Now what we want is to create a model to work upon so that we can feed the neural network some input to train.

Keras provides various libraries to create a model for our neural network. These models provide us to a decent technique to organize layers. Here we will be building a simple sequential method which will comprise of three layers.

Here each layer would have different number of neurons depending upon the input we want, the number of output classes we expect and also the degree of learning. We know the number of inputs we have and we also know the number of classes to which each image can be classified, now we must cogitate over the number of neurons the hidden layer must have. One general rule for selecting the number of neurons in the hidden layer is that the number must not be either too large or close to the number of inputs because then the network would be very hard to train on the other hand if we choose a number that is too small or close to the number of output classes then we will not have the ability to learn.

Now we know that each data entry corresponds to 784 inputs and we also know that there are only 10 output classes in which these images can be classified. Hence, the number of neurons in the first layer (input layer) will be 784 and the number of neurons in the third layer (output layer) will be equal to 10. The number of neurons that the second layer (hidden layer) must have will lie between 10 and 784. Considering the degree of learning we want to provide to the network we selected 200 neurons which would be theoretically appropriate.

```
model.add(Dense(784, input_dim=784, activation='relu',init='uniform'))
model.add(Dense(200, activation='relu',init='uniform'))
model.add(Dense(num_classes, activation='sigmoid',init='uniform'))
```

Fig.2. add() function

Hitherto, we have a basic idea of the structure of the neural network we will be constructing, but the things get a little more complex when we actually code the ANN. Keras provides us with some core layers that helps us to stack various individual layers onto each other. We will be using Dense Layer which will create a regular densely-connected NN layer and add the three (input, hidden and the output) layers while setting various parameters such as ‘number of neurons’, ‘input dimension’ , ‘initializers’ and ‘activation function’.

We know that the neural network works by raising the weight associated with the connection between two connected nodes. Larger the weight larger the amplification of the signal passing through it. These weights will keep on adjusting while we train the network. For initializing the weights of these connections before starting the training phase we utilize the “uniform distribution” to allocate the weights randomly. The premise of choosing a random number can be justified by the fact that while training against the dataset these weights will either be increased or decreased repeatedly, it will not matter much how you initialize the network.

C. ACTIVATION FUNCTION

Activation functions are the core of the neural network in that they decide whether the strength of the summation of all the inputs to a given node is sufficient enough to fire the output. These functions decide the sensitivity of each neuron. We will be using ‘Rectifier function (Relu)’ in our first two layers and ‘sigmoid function’ in our final layer. Rectifier function has proven itself to provide an edge over other function which means better predictions. We used sigmoid function in the output layer to provide an output in the range of 0 and 1 that’s what we want. In our experiment, there are ten classes in which each image can be classified, and for knowing which class has been predicted we will compare the output of each class which will lie between 0 and 1. The class having the largest value relative to the other classes will be considered as the prediction[13].

D. LEARNING PHASE

To this point we have added the three layers to our model, we now require to compile the model. Compiling a network means specifying various attributes of the process of learning. The learning phase largely governs the performance of the neural network because if the strength of the connections among the nodes is not properly defined then it will not matter which type of model you created. Hence, we will be focusing on this phase by varying various attributes and observe the impact made by each change.

```
model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

Fig.4 compile() function

Loss function and optimizer are the two attributes or parameters associated with the compilation of the model. The loss function calculates the 'error' or difference between the desired output and the output of the network for each training example. Rather than updating weights after each training example, what we do is divide the dataset into batches and average the error of each batch, this error is then used to update the weights. Optimizers on the other hand are algorithms whose task is to reduce this error. The optimizer and the loss function work side by side to evaluate the error and progressively reduce that error to reach the optimal output. Keras provides us with various loss functions and optimizers to choose from.

E. EXECUTION PHASE

In the previous phase we compiled the model, now we will be executing the model. Keras provides us with a function fit() which allows us to set parameters to the execution phase. The two parameters that are worth the attention are nb_epoch and batch_size.

```
model.fit(Input, Output, nb_epoch=10, batch_size=20)
```

Fig.3 fit() function

The parameter, nb_epoch represents the number of iterations that the model must go through over the entire dataset for training the network. The batch_size parameter represents after how many instance of images the weights must be updated.

F. EXECUTION PHASE

Now we must evaluate the performance of each model, for doing so we will be using the training dataset as our testing dataset which means after training the network on the dataset we created we will evaluate the network using the same dataset. Keras provides us with a function evaluate() to evaluate the model which takes two arguments input (image data) and output (desired classification)

```
scores = model.evaluate(Input, Output)
```

Fig.4 evaluate() function

VI.EXPERIMENTATION

Here, in this phase we will be experimenting with the parameters of the learning phase and try to learn the impact of each change in parameter. We will be selecting categorical_crossentropy and binary_crossentropy as our loss functions and using them one at a time with each optimizer. We will be using adam and adadelta as the optimizers as they are known for their effectiveness and efficiency.

1) MODEL 1

Loss function: categorical_crossentropy optimizer: adadelta Accuracy: 78.89

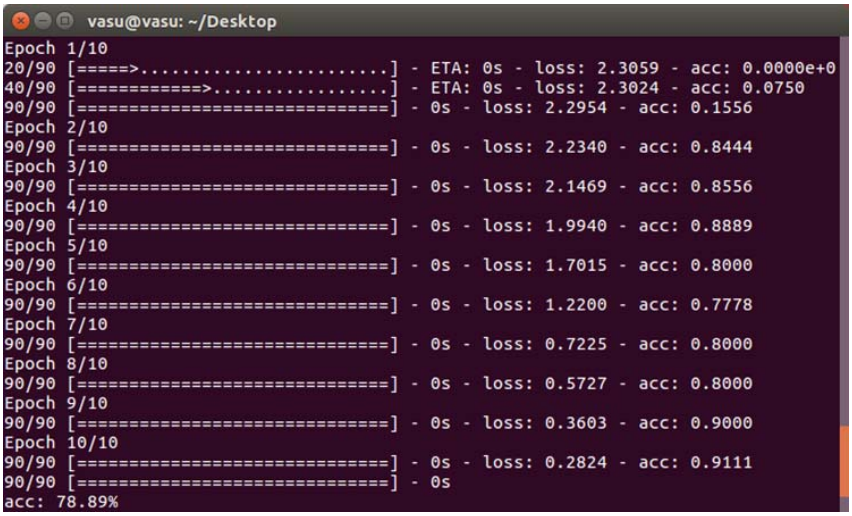

Fig.5 Model -1

2) MODEL 2

Loss function: binary_crossentropy optimizer: adadelta Accuracy: 90

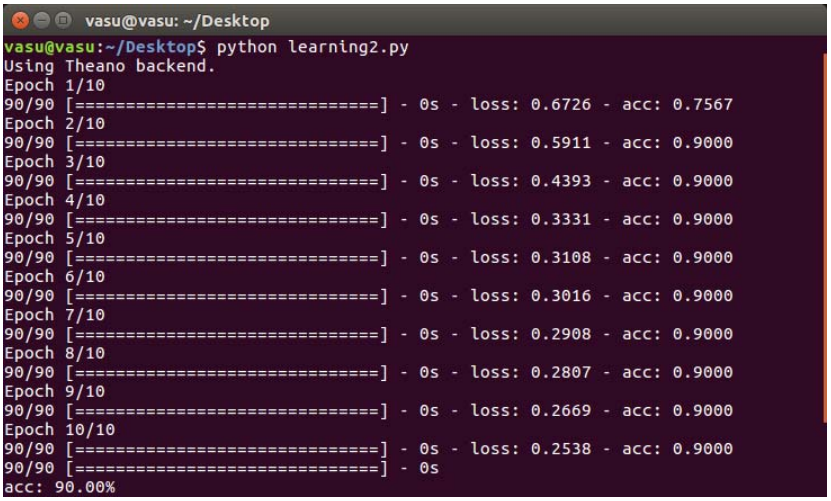

Fig.6 Model -2

3) MODEL 3

Loss function: binary_crossentropy optimizer: adam Accuracy: 99.67

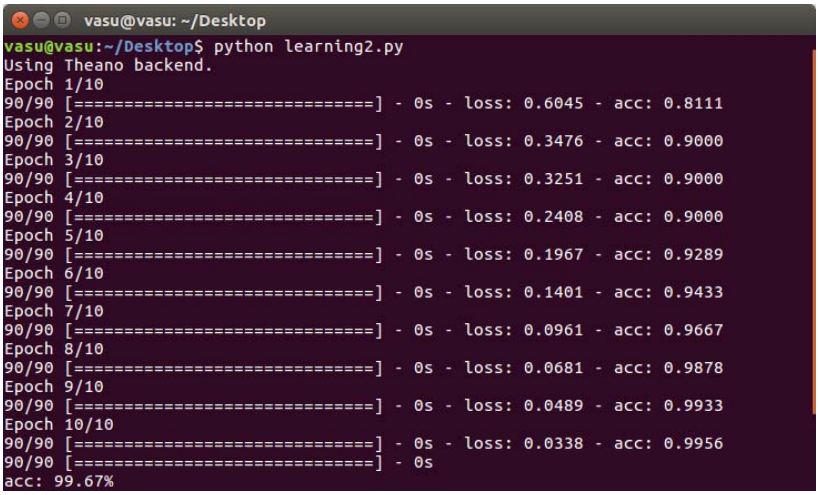

Fig.7 Model -3

4) MODEL 4

Loss function: categorical _crossentropy optimizer: adam Accuracy: 100

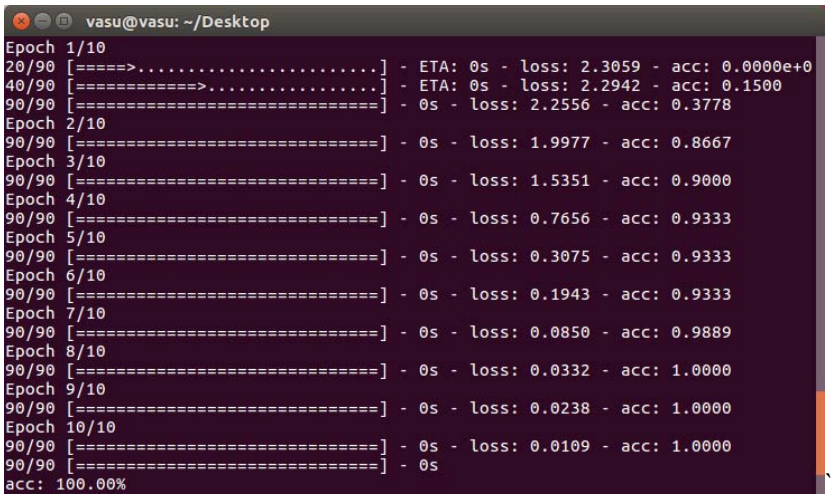

Fig.8 Model -4

VII. OBSERVATION

From the above trials, we were able to observe that categorical_crossentropy and adam when used in combination gives better results than the other combinations. The other combination which was also impressive was binary_crossentropy and adam which gave an accuracy of 99.67%. We were able to achieve this high performance because there was no noise in the input during the evaluation phase, in other words as we used the same dataset for evaluating the performance of each model as the one used in training our network and that's why no new variation in the input of each class was introduced due to which the network is able to effectively classify the characters. But in this way we were able to effectively juxtapose the models with each other and carefully evaluate the performance of each model with respective to each other.

VIII. CONCLUSION AND FUTURE SCOPE

We found out that neural network can prove to be effective and accurate for the task of recognizing the Devanagari characters. The impressive library of Keras and Theano helped in the construction of complex network and also provided with various parameters to finely tune the network. We observed that the performance of an artificial neural network is greatly influenced by the choice of parameters of the learning phase and disregarding the importance of this phase will greatly hamper with the accuracy. While we were experimenting with the parameters of the learning phase, we found out that the different combinations of loss function and optimizer showed large deviation in the performance of the network. We mix-matched between two loss functions and two optimizers taking only one at a time and tried to find out the variation in the accuracy. The combination of categorical_crossentropy and adam which we selected as our loss function and optimizer outperformed other combinations on the other hand the combination of categorical_crossentropy and adadelta had the worst performance with a difference of 21.11 percent in accuracy when compared to the previous combination. This shows the importance of choosing the correct attributes of the learning phase and how much the accuracy may vary by choosing incompatible attributes for our network.Our future work will comprise of experimenting with various other attributes associated with different phases and analysing the effect of each change.

REFERENCES

[1] George Cardona and Danesh Jain (2003), The Indo-Aryan Languages, Routledge, ISBN 978-0415772945, pages 68-69
[2] Fabian Pedregosa, Gaël, Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay, " Scikit-learn: Machine Learning in Python", The Journal of Machine Learning Research, Volume 12, 2/1/2011, Pages 2825-2830
[3] B Yegnanarayana ,Artificial neural networks, PHI Learning Pvt. Ltd
[4] Hebb, Donald (1949). The Organization of Behavior. New York: Wiley.
[5] Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review. 65 (6): 386–408.
[6] Werbos, P.J. (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences
[7] PL Bartlett,"The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network", IEEE transactions on Information Theory 44 (2), 525-536
[8] V Koval, V Turchenko, V Kochan, A Sachenko, G Markowsky,"Smart license plate recognition system based on image processing using neural network", 2003
[9] Md. Iqbal Quraishi, J Pal Choudhury, Mallika De,"Image recognition and processing using Artificial Neural Network", 2012
[10] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E, "ImageNet Classification with Deep Convolutional Neural Networks", 2012
[11] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, Yoshua Bengio, Theano: A CPU and GPU Math Compiler in Python ,2010
[12] François Chollet, "Keras",(2013), GitHub repository, https://github.com/fchollet/keras
[13] Tariq Rashid, Make Your own neural network, CreateSpace Independent Publishing Platform, 2016